Stock Market Analysis Tool

By: Pranav Menon, Fall 2022

In this tutorial, we will be exploring the various stages of the data science pipeline, including data collection and parsing, data management and representation, exploratory data analysis, hypothesis testing, and communication of insights. By the end of this tutorial, readers should have a better understanding of the data science process and be able to apply these techniques to their own datasets. Our tutorial will be presented in the form of a Jupyter Notebook, using a combination of Markdown and Python code to clearly demonstrate each step of the process. We will be using a live stock dataset from yahoo finance for the purpose of this tutorial, but the techniques and approaches discussed can be applied to any dataset.

This is a tutorial on how to create a stock market analysis tool using the Python programming language and the yfinance library. The tutorial will be presented in the form of a Jupyter Notebook, using a combination of Markdown and Python code to clearly demonstrate each step of the process. The tutorial will use a live dataset of the stock market from Yahoo Finance, but the techniques and approaches discussed can be applied to any dataset. The tutorial will cover the analysis of four tech stocks: TESLA, NVIDIA, WALMART, and META. The tutorial will show how to download and analyze the data, create graphs to visualize the data, and calculate statistical measures such as percent change and correlation. (To run this code, make sure to run the following two pip installations at the beginning of your code)

In [247]:

```
# make sure to install the yfinance library using this command to utilize the updated yah
oo finance data
!pip install yfinance
!pip install fix_yahoo_finance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Requirement already satisfied: yfinance in /usr/local/lib/python3.8/dist-packages (0.1.90
)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.8/dist-packages (
from yfinance) (1.4.4)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.8/dist-packa
ges (from yfinance) (0.0.11)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.8/dist-packages (
from yfinance) (2.28.1)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-packages (fro
```

```
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-packages (fro
m yfinance) (4.9.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.8/dist-packages (f
rom yfinance) (1.21.6)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.8/dist-packages (f
rom yfinance) (1.3.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-pa
ckages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (fr
om pandas>=1.3.0->yfinance) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from p
ython-dateutil>=2.7.3->pandas>=1.3.0->yfinance) (1.15.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-pac
kages (from requests>=2.26->yfinance) (1.24.3)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.8/dist-
packages (from requests>=2.26->yfinance) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packag
es (from requests>=2.26->yfinance) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-packages (fr
om requests>=2.26->yfinance) (2.10)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Requirement already satisfied: fix_yahoo_finance in /usr/local/lib/python3.8/dist-package
s (0.0.22)
Requirement already satisfied: multitasking in /usr/local/lib/python3.8/dist-packages (fr
om fix_yahoo_finance) (0.0.11)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from fix_
yahoo_finance) (1.21.6)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from fix
_yahoo_finance) (1.3.5)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from f
ix_yahoo_finance) (2.28.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (fr
om pandas->fix_yahoo_finance) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-pa
ckages (from pandas->fix_yahoo_finance) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from p
ython-dateutil>=2.7.3->pandas->fix_yahoo_finance) (1.15.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-packages (fr
om requests->fix_yahoo_finance) (2.10)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.8/dist-
packages (from requests->fix_yahoo_finance) (2.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-pac
kages (from requests->fix_yahoo_finance) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packag
es (from requests->fix_yahoo_finance) (2022.12.7)
```

To start, we will begin by importing the necessary libraries and specifying the tech stocks that we will be analyzing in this tutorial. The pandas library will be used for data manipulation and analysis, numpy for numerical computation, matplotlib and seaborn for data visualization, and datetime for handling dates and times. We will also be using the yfinance library to download stock data from Yahoo Finance.

In [248]:

```
# the imports I am using
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
# For reading stock data from yahoo
from pandas_datareader.data import DataReader
import yfinance as yahoostocks
```

Next, we will specify the tech stocks that we will be analyzing in this tutorial. We will be using the ticker symbols for TESLA (TSLA), NVIDIA (NVDA), WALMART (WMT), and META (META). We will also specify the start and end dates for the analysis, which will be one year from the current date. We will then use the yfinance library to download the stock data for the specified companies and dates. The stock data will be stored in a separate Pandas DataFrame for each company.

In [249]:

```
# The tech stocks we'll use for this analysis
companies = ['TSLA', 'NVDA', 'WMT', 'META']

# End and start times so that getting the times from data frame will be simple
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

#downloading the specified stocks from yf
for stock in companies:
    globals()[stock] = yahoostocks.download(stock, start, end)
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
```

We will create some lists with the stock names which we will use frequently to call the names of the companies and their corresponding data. We will then add the names of the companies to each DataFrame as a new column. We can then concatenate the DataFrames into a single DataFrame using the pd.concat() function.

In [250]:

```
#creating some lists with the stock names which we will use frequently to call the names
companieslist = [TSLA, NVDA, WMT, META]
companiesname = ['TESLA', 'NVIDIA', 'WALMART', 'META']

#adding all of the data to a dataframe
for company, nextcom in zip(companieslist, companiesname):
    company["company_name"] = nextcom

stockframe1 = pd.concat(companieslist)
stockframe1
```

Out[250]:

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2021-12-17	304.923340	320.220001	303.013336	310.856659	310.856659	100437300	TESLA
2021-12-20	303.566681	307.230011	297.796661	299.980011	299.980011	56480100	TESLA

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2021-12-21	305.623322	313.166656	295.373322	312.843323	312.843323	71517000	TESLA
2021-12-22	321.886658	338.553345	319.016663	336.290009	336.290009	93634200	TESLA
2021-12-23	335.600006	357.660004	332.519989	355.666656	355.666656	92713200	TESLA
...	...	...	...	...	...	...	...
2022-12-12	115.180000	115.720001	113.139999	114.709999	114.709999	24747100	META
2022-12-13	122.129997	123.300003	118.639999	120.150002	120.150002	44701100	META
2022-12-14	119.389999	124.139999	119.389999	121.589996	121.589996	36922000	META
2022-12-15	118.330002	118.629997	114.010002	116.150002	116.150002	34531000	META
2022-12-16	120.230003	123.309998	118.820000	119.430000	119.430000	67045000	META

1008 rows × 7 columns

Next, we will generate some descriptive statistics and information about each stock using the describe() and info() methods. This will give us a general understanding of the structure and content of the data.

In [251]:

```
# Telsa descriptions of the data frame.
telsadf = TSLA.describe()
print(TSLA.info())
telsadf
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2021-12-17 to 2022-12-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          252 non-null    float64
 1   High          252 non-null    float64
 2   Low           252 non-null    float64
 3   Close         252 non-null    float64
 4   Adj Close     252 non-null    float64
 5   Volume        252 non-null    int64  
 6   company_name  252 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ KB
None
```

Out[251]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	271.845701	278.548770	264.237460	271.071521	271.071521	8.285525e+07
std	53.505444	54.462109	52.397512	53.473644	53.473644	2.286666e+07
min	153.440002	160.929993	150.039993	150.229996	150.229996	4.073370e+07
25%	230.709999	238.340000	223.962498	232.554996	232.554996	6.658005e+07
50%	279.946671	284.294998	270.511658	276.763336	276.763336	8.017800e+07
75%	304.695824	312.415840	299.183327	306.685829	306.685829	9.537915e+07
max	396.516663	402.666656	378.679993	399.926666	399.926666	1.758627e+08

In [252]:

```
# Nvidia descriptions of the data frame.
nvidiadf = NVDA.describe()
print(NVDA.info())
nvidiadf
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2021-12-17 to 2022-12-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
```

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          252 non-null    float64
 1   High          252 non-null    float64
 2   Low           252 non-null    float64
 3   Close         252 non-null    float64
 4   Adj Close     252 non-null    float64
 5   Volume        252 non-null    int64
 6   company_name  252 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ KB
None
```

Out[252]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	191.216547	195.974603	186.332342	191.188452	191.065176	5.437910e+07
std	50.774593	51.759209	49.131516	50.416782	50.341805	1.327516e+07
min	109.709999	117.349998	108.129997	112.269997	112.241280	1.679340e+07
25%	156.174995	159.395000	151.790005	156.294998	156.243446	4.570548e+07
50%	178.350006	181.419998	171.894997	178.280006	178.172600	5.243335e+07
75%	234.129997	240.587505	226.115005	233.725002	233.506657	6.310748e+07
max	313.119995	313.299988	300.119995	309.450012	309.160919	1.178865e+08

In [253]:

```
# Walmart descriptions of the data frame.
walmartdf = WMT.describe()
print(WMT.info())
walmartdf
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2021-12-17 to 2022-12-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          252 non-null    float64
 1   High          252 non-null    float64
 2   Low           252 non-null    float64
 3   Close         252 non-null    float64
 4   Adj Close     252 non-null    float64
 5   Volume        252 non-null    int64
 6   company_name  252 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ KB
None
```

Out[253]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	138.065912	139.543968	136.838413	138.168334	136.957720	8.111334e+06
std	10.019886	9.923452	9.962776	9.925810	9.753030	4.702258e+06
min	118.300003	120.000000	117.269997	118.290001	117.334724	2.925800e+06
25%	131.247498	132.940002	129.972496	131.542503	130.934536	5.856250e+06
50%	137.955002	139.669998	137.050003	138.030006	136.549133	6.974200e+06
75%	144.457497	145.612495	143.134995	144.919998	142.766262	8.736925e+06
max	160.250000	160.770004	159.070007	159.869995	158.004593	4.431340e+07

In [254]:

```
# Meta descriptions of the data frame.
metadf = META.describe()
print(META.info())
metadf
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2021-12-17 to 2022-12-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          252 non-null    float64
 1   High          252 non-null    float64
 2   Low           252 non-null    float64
 3   Close         252 non-null    float64
 4   Adj Close     252 non-null    float64
 5   Volume        252 non-null    int64  
 6   company_name  252 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ KB
None
```

Out[254]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	188.744167	192.417738	185.305834	188.661509	188.661509	3.529868e+07
std	63.098457	63.807571	62.183457	63.034347	63.034347	2.400893e+07
min	90.080002	90.459999	88.089996	88.910004	88.910004	1.059330e+07
25%	146.237499	148.774998	144.897503	146.239994	146.239994	2.318452e+07
50%	177.125000	181.100006	174.595001	176.635002	176.635002	2.948040e+07
75%	211.482498	216.657501	207.745003	211.625004	211.625004	3.890492e+07
max	346.910004	352.709991	345.200012	346.220001	346.220001	2.323166e+08

We will then use matplotlib to create graphs visualizing the volume of each stock over time. We will use a for loop to iterate over the list of stock DataFrames and create a separate graph for each one. The subplot() function will be used to create a 2x2 grid of plots, with one plot for each stock. The plot() function will be used to draw the line graphs, with the volume data plotted in green and the adjusted closing price plotted in purple.

The volume data for the four companies (Tesla, NVIDIA, Walmart, and Meta) provides information about the number of shares traded in a particular stock on a given day. This data can be useful for identifying trends in trading activity for a particular stock and for comparing the relative popularity or liquidity of different stocks.

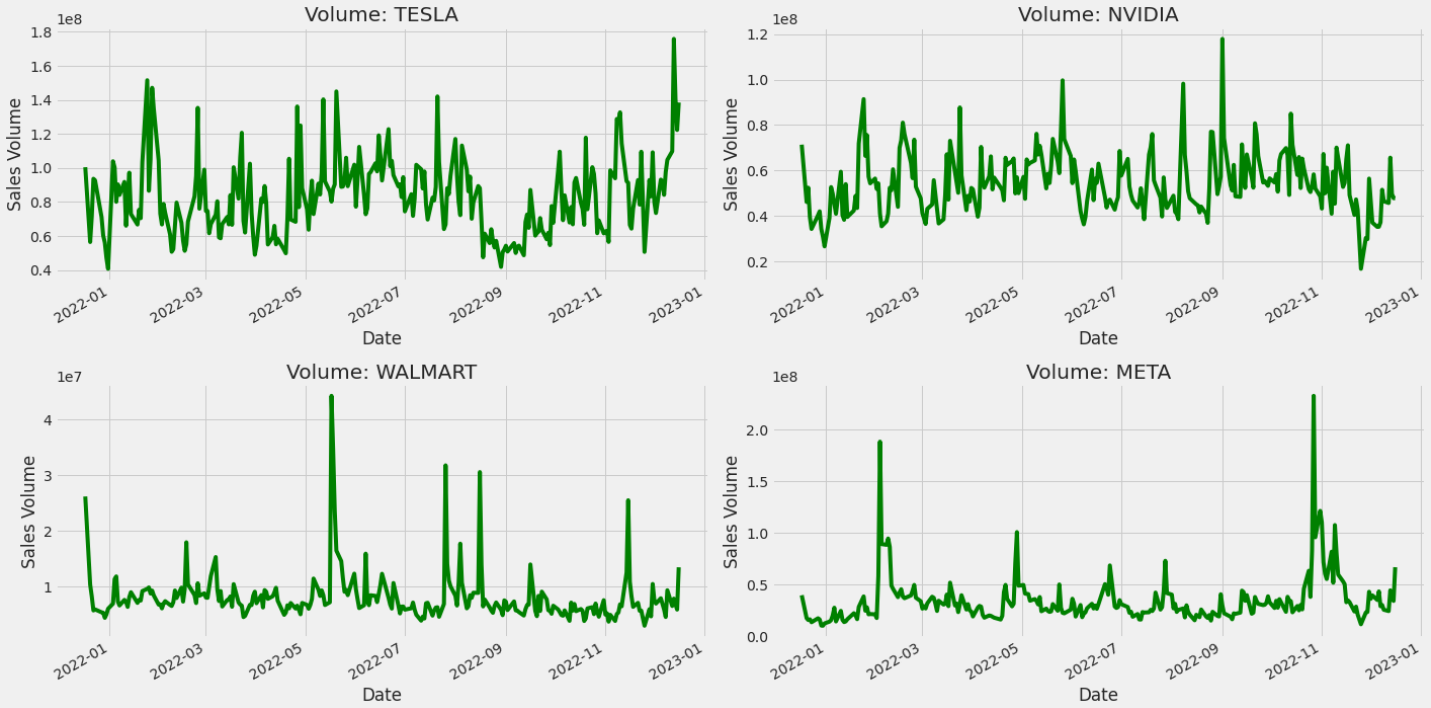
The volume data for each of the four companies is plotted on a separate graph, with the x-axis representing the date and the y-axis representing the volume traded. By examining the volume data for each company, it is possible to see how the trading activity for that stock has changed over time. For example, if the volume for a particular stock is consistently increasing, it could indicate that there is growing interest in the stock and that it is becoming more popular with investors. On the other hand, if the volume is consistently decreasing, it could indicate that the stock is becoming less popular or less liquid.

It is also possible to compare the volume data for the different companies in order to see how they compare in terms of trading activity. For example, if one company's volume is consistently higher than the others, it could indicate that the stock is more popular or more liquid than the other stocks. Additionally, it is possible to examine whether there are any correlations between the volume data for different companies. For example, if the volume for two companies tends to increase or decrease at the same time, it could suggest that the two stocks are related or that the trading activity for one stock is influencing the other.

In [255]:

```
#setting the size of plt
plt.figure(figsize=(20, 10))
```

```
#creating a graph for each stock for the closing price
for i, company in enumerate(companieslist, 1):
    plt.subplot(2, 2, i)
    plt.ylabel('Sales Volume')
    plt.xlabel('Date')
    #plot volume with color green
    company['Volume'].plot(color='green')

    plt.title("Volume: " + companiesname[i - 1])

plt.tight_layout()
```



We will then use matplotlib to create graphs visualizing the adjusted closing price of each stock over time. We will use a for loop to iterate over the list of stock DataFrames and create a separate graph for each one. The subplot() function will be used to create a 2x2 grid of plots, with one plot for each stock. The plot() function will be used to draw the line graphs, with the volume data plotted in green and the adjusted closing price plotted in purple.

The closing price data for the four companies (Tesla, NVIDIA, Walmart, and Meta) provides information about the price at which a particular stock closed on a given day. This data can be useful for identifying trends in the price of a particular stock and for comparing the relative performance of different stocks.

The closing price data for each of the four companies is plotted on a separate graph, with the x-axis representing the date and the y-axis representing the closing price. By examining the closing price data for each company, it is possible to see how the price of that stock has changed over time. For example, if the closing price for a particular stock is consistently increasing, it could indicate that the stock is performing well and that investors are confident in the company. On the other hand, if the closing price is consistently decreasing, it could indicate that the stock is underperforming or that there is negative sentiment surrounding the company.

It is also possible to compare the closing price data for the different companies in order to see how they compare in terms of performance. For example, if one company's closing price is consistently higher than the others, it could indicate that the stock is performing better or is more valuable than the other stocks. Additionally, it is possible to examine whether there are any correlations between the closing price data for different companies. For example, if the closing price for two companies tends to increase or decrease at the same time, it could suggest that the two stocks are related or that the performance of one stock is influencing the other.

In [256]:

```
#setting the size of plt
plt.figure(figsize=(20, 10))

#creating a graph for each stock for the closing price
```

```
for i, company in enumerate(companieslist, 1):
    plt.subplot(2, 2, i)
    plt.ylabel('Adjusted Closing Price')
    plt.xlabel('Date')
    #plot adjusted close price with color purple
    company['Adj Close'].plot(color='purple')

    plt.title("Closing Price: " + companiesname[i - 1])

plt.tight_layout()
```

The code first creates a figure with a 2x2 grid of subplots using matplotlib. It then calculates the daily percentage gain for each stock by using the pct_change() method on the Adj Close column of each stock DataFrame. It then plots the daily return data for each stock in a separate subplot, using the plot() function and specifying the x and y axes, the legend, the linestyle, the marker, and the color. The set_title() method is used to label each subplot with the name of the corresponding stock.

The daily percentage gain data for the four companies (Tesla, NVIDIA, Walmart, and Meta) provides information about the percentage change in the price of a particular stock from one day to the next. This data can be useful for identifying trends in the price of a particular stock and for comparing the relative performance of different stocks on a daily basis.

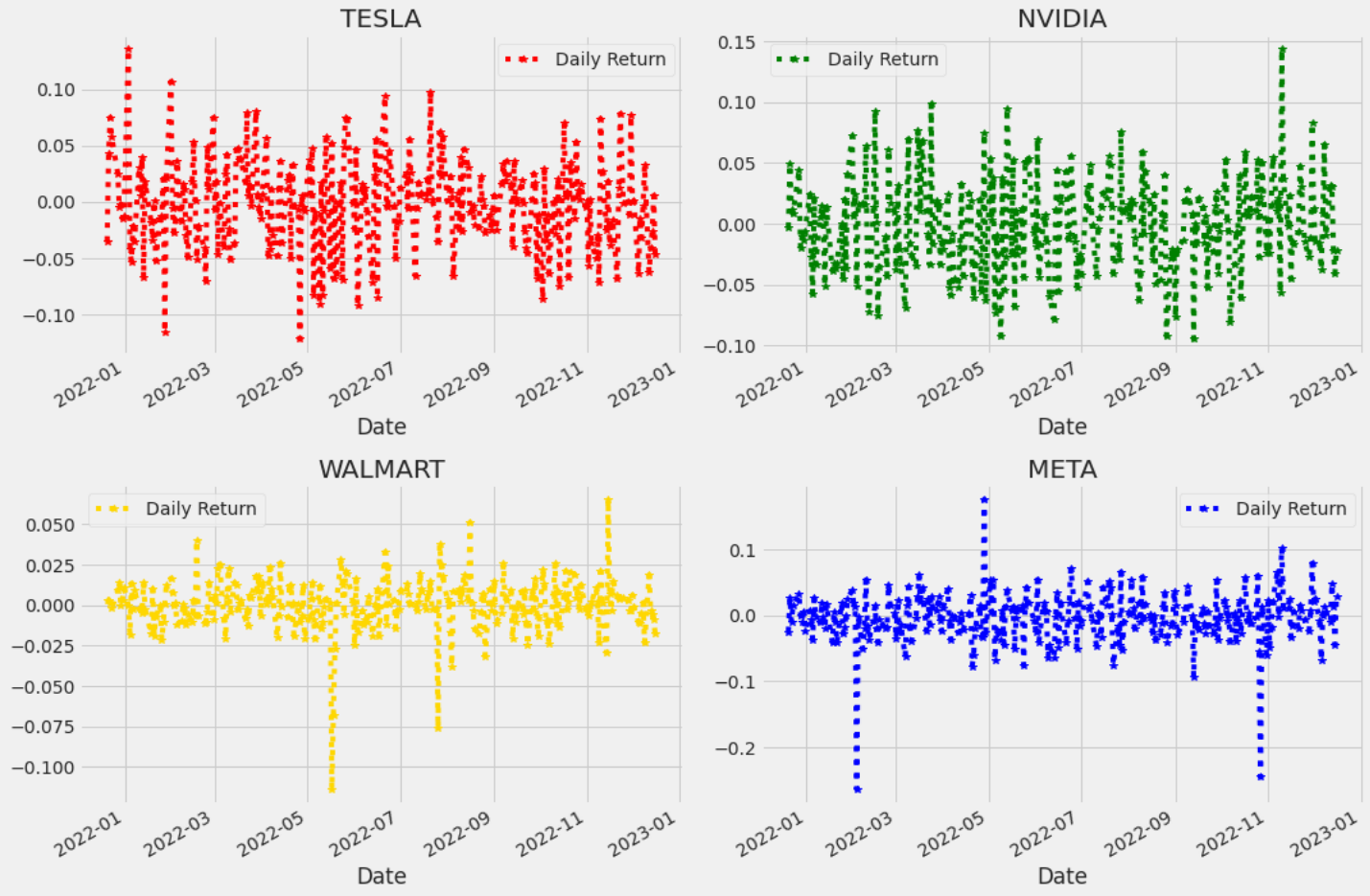
The daily percentage gain data for each of the four companies is plotted on a separate graph, with the x-axis representing the date and the y-axis representing the percentage change in the stock's price. By examining the daily percentage gain data for each company, it is possible to see how the price of that stock has changed on a daily basis. For example, if the daily percentage gain for a particular stock is consistently positive, it could indicate that the stock is performing well and that investors are confident in the company. On the other hand, if the daily percentage gain is consistently negative, it could indicate that the stock is underperforming or that there is negative sentiment surrounding the company.

It is also possible to compare the daily percentage gain data for the different companies in order to see how they compare in terms of performance on a daily basis. For example, if one company's daily percentage gain is consistently higher than the others, it could indicate that the stock is performing better or is more valuable than the other stocks on a daily basis. Additionally, it is possible to examine whether there are any correlations between the daily percentage gain data for different companies. For example, if the daily percentage gain for two companies tends to increase or decrease at the same time, it could suggest that the two stocks are related or that the performance of one stock is influencing the other on a daily basis.

In [257]:

```
# graph specifications
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
```

```
fig.set_figwidth(15)

# looping through the companies again
for stock in companieslist:
    #this is to find the percent change daily
    stock['Daily Return'] = stock['Adj Close'].pct_change()

# daily percentage gain for each of the stocks
TSLA['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='dotted', marker='*', col
or = 'red')
axes[0,0].set_title('TESLA')

NVDA['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='dotted', marker='*', col
or = 'green')
axes[0,1].set_title('NVIDIA')

WMT['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='dotted', marker='*', colo
r = 'gold')
axes[1,0].set_title('WALMART')

META['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='dotted', marker='*', col
or = 'blue')
axes[1,1].set_title('META')

fig.tight_layout()
```



Next, the code creates another figure and uses a for loop to create histograms of the daily return data for each stock. It uses the hist() function to plot the data, specifying the number of bins and the color of the bars. It also uses the ylabel() and xlabel() functions to label the y-axis and x-axis, respectively. The tight_layout() function is used to ensure that the plots do not overlap.

The average daily return data, represented by histograms in the provided code, provides information about the distribution of daily percentage gains for a particular stock over a given period of time. This data can be useful for understanding the risk and volatility of a particular stock, as well as for comparing the relative risk and volatility of different stocks.

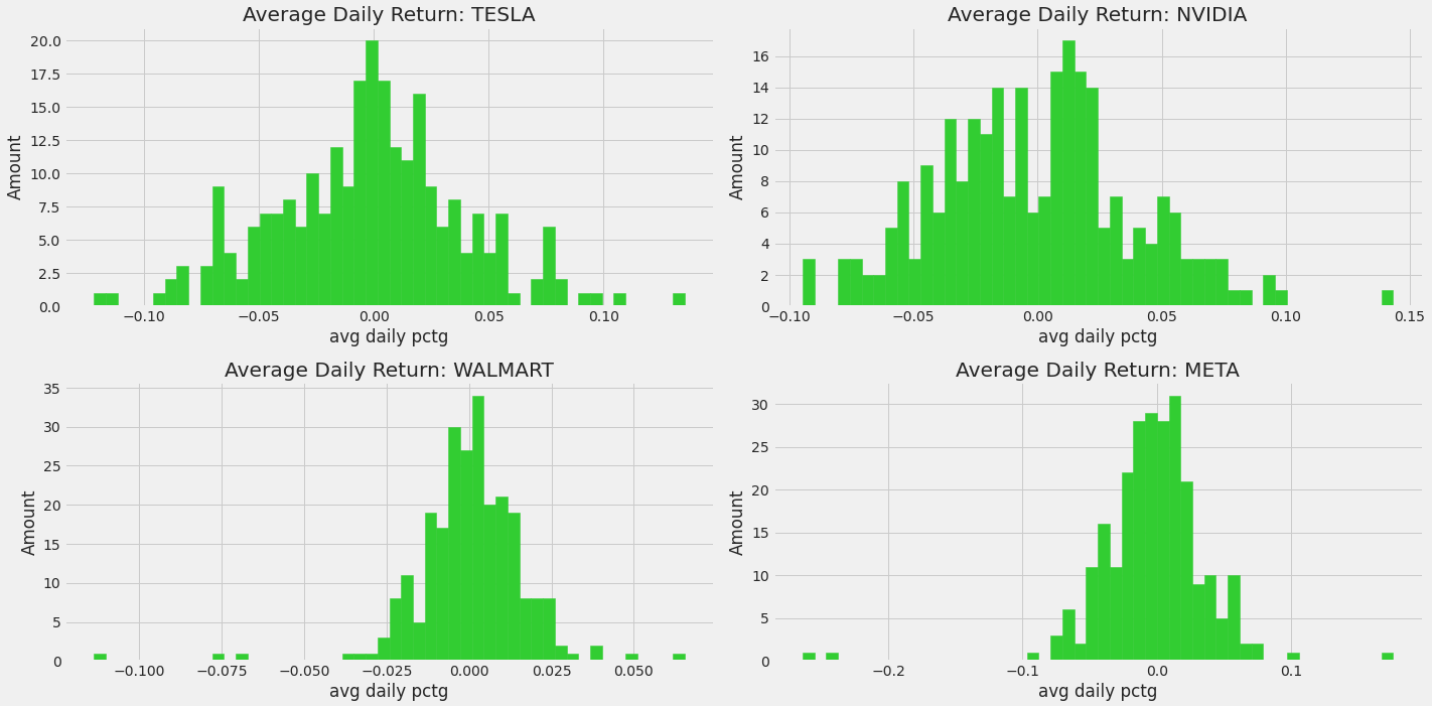
Histograms are generated for the average daily return data of each of the four companies (Tesla, NVIDIA, Walmart, and Meta). These histograms show the frequency of different daily percentage gains for the stock, with

the x-axis representing the percentage gain and the y-axis representing the frequency of that percentage gain. By examining the shape of the histogram, it is possible to get a sense of the overall distribution of the daily percentage gains for the stock. For example, if the histogram is skewed to the left or right, it could indicate that the stock is more prone to extreme percentage gains (either positive or negative) and is therefore more volatile. On the other hand, if the histogram is more symmetrical, it could suggest that the stock is more stable and has a more even distribution of daily percentage gains.

It is also possible to compare the average daily return data for the different companies in order to see how they compare in terms of risk and volatility. For example, if one company's histogram is skewed significantly to the left or right, it could indicate that the stock is more volatile than the other stocks. Additionally, it is possible to examine whether there are any correlations between the average daily return data for different companies. For example, if the histograms for two companies are similarly shaped, it could suggest that the two stocks are similarly risky or volatile.

In [258]:

```
# graph specifications
plt.figure(figsize=(20, 10))

#creating a graph for each stock for the closing price
for i, company in enumerate(companieslist, 1):
    plt.subplot(2, 2, i)
    plt.title("Average Daily Return: " + companiesname[i - 1])
    company['Daily Return'].hist(bins=50, color = "limegreen", ec="limegreen")
    plt.ylabel('Amount')
    plt.xlabel('avg daily pctg')

plt.tight_layout()
```

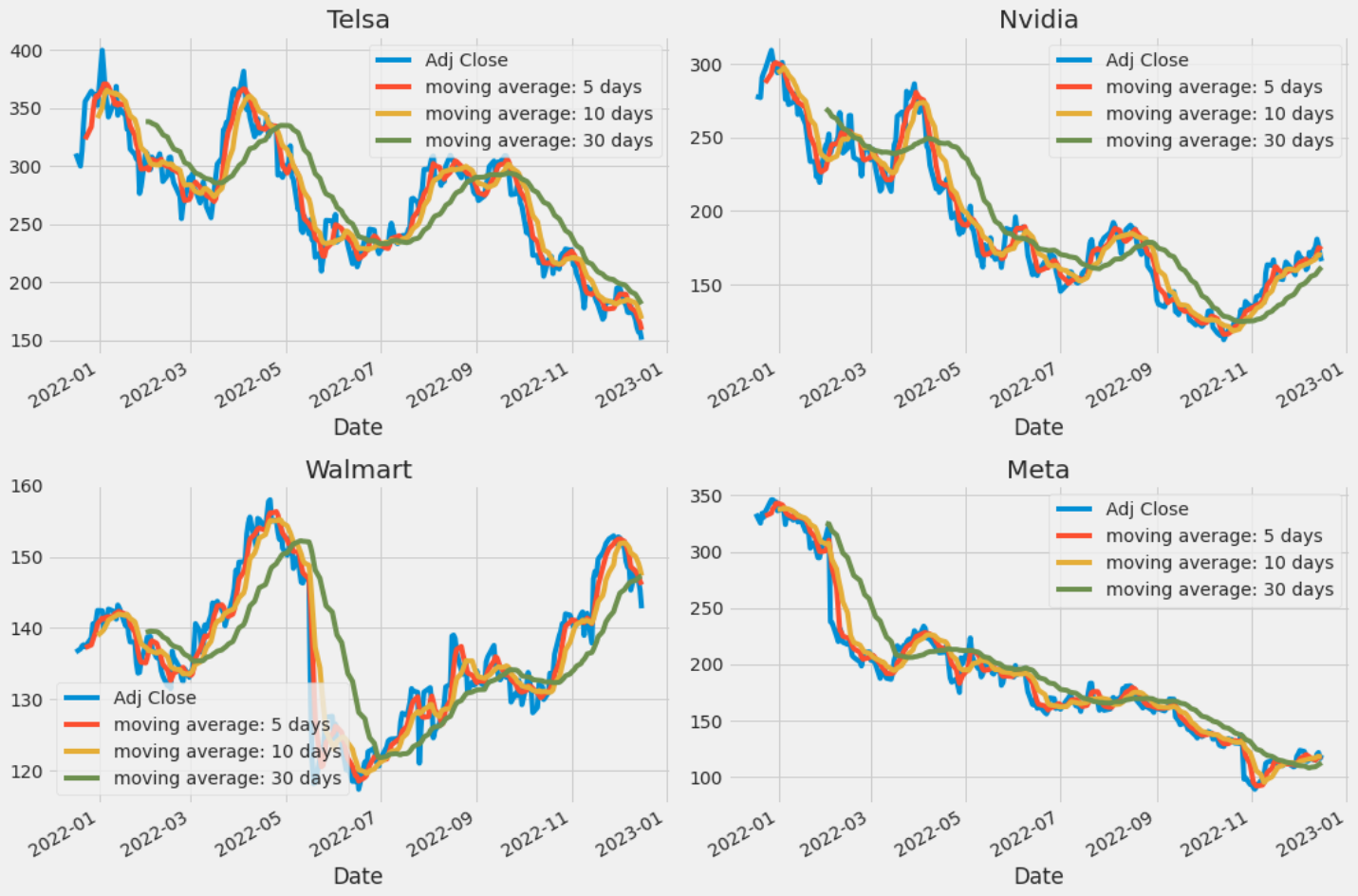


The code then creates another figure with a 2x2 grid of subplots and defines three moving average periods (5 days, 10 days, and 30 days). It uses a nested for loop to iterate over the list of stock DataFrames and the list of moving average periods. For each combination, it calculates the rolling mean using the rolling() method and stores the result in a new column of the DataFrame. It then plots the Adj Close data and the moving average data for each stock in a separate subplot, using the plot() function and specifying the x and y axes. The tight_layout() function is used again to ensure that the plots do not overlap.

In [259]:

```
# graph specifications
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

# creating certain days for the moving average to analyze by.
```

```
movingavg = [5, 10, 30]
# for loop to go through the data frame
for i in movingavg:
    for company in companieslist:
        #creating a column with the rolling mean claculations for moving average for each
stock
        column_name = f"moving average: {i} days"
        company[column_name] = company['Adj Close'].rolling(i).mean()

#moving average (5, 10, 30 days) compared to the adj close of each stock
TSLA[['Adj Close', 'moving average: 5 days', 'moving average: 10 days', 'moving average:
30 days']].plot(ax=axes[0,0])
axes[0,0].set_title('Telsa')

NVDA[['Adj Close', 'moving average: 5 days', 'moving average: 10 days', 'moving average:
30 days']].plot(ax=axes[0,1])
axes[0,1].set_title('Nvidia')

WMT[['Adj Close', 'moving average: 5 days', 'moving average: 10 days', 'moving average:
30 days']].plot(ax=axes[1,0])
axes[1,0].set_title('Walmart')

META[['Adj Close', 'moving average: 5 days', 'moving average: 10 days', 'moving average:
30 days']].plot(ax=axes[1,1])
axes[1,1].set_title('Meta')

fig.tight_layout()
```



Finally, the code downloads the Adj Close data for the stocks again from Yahoo Finance, but this time it reorders the data to change the row and column format for analyzing the Adj Close data. It creates a new DataFrame with all of the Adj Close data and calculates the daily percentage gain for each stock using the pct_change() method. It then prints the resulting data frame to the console.

In [260]:

```
#downloading the stock information from yahoo again but this time
# reordering the information to change the row and col format for analyzing Adj Close
adjclosedj = yahoostocks.download(companies, start = "2019-01-01", end ="2022-12-15", gro
up_by = "ticker")
```

```
adjclosedj = adjclosedj.xs('Adj Close', axis = 1, level = 1, drop_level = True)

# new data frame with all information about the adj return
adjfinaldf = adjclosedj.pct_change()
adjfinaldf
```

```
[**********100%**********]  4 of 4 completed
```

Out[260]:

Date	NVDA	META	TSLA	WMT
2019-01-02	NaN	NaN	NaN	NaN
2019-01-03	-0.060417	-0.029039	-0.031472	-0.005142
2019-01-04	0.064067	0.047138	0.057697	0.006246
2019-01-07	0.052941	0.000725	0.054361	0.011772
2019-01-08	-0.024895	0.032452	0.001164	0.006981
...	...	...	...	...
2022-12-08	0.065074	0.012288	-0.003447	0.001548
2022-12-09	-0.009785	0.004942	0.032345	-0.023323
2022-12-12	0.031410	-0.010267	-0.062720	0.018650
2022-12-13	0.030624	0.047424	-0.040937	-0.003581
2022-12-14	-0.022023	0.011985	-0.025784	-0.005560

997 rows × 4 columns

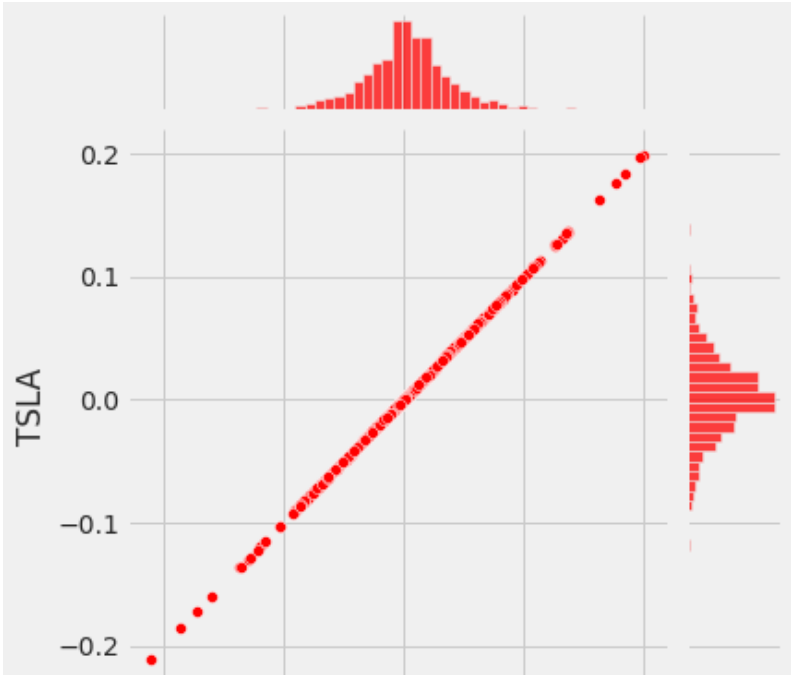
The code first creates four seaborn joint plots to visualize the relationship between each stock's Adj Close data and itself. It uses the jointplot() function and specifies the x-axis and y-axis data, the data source, the type of plot, and the color.

In [261]:

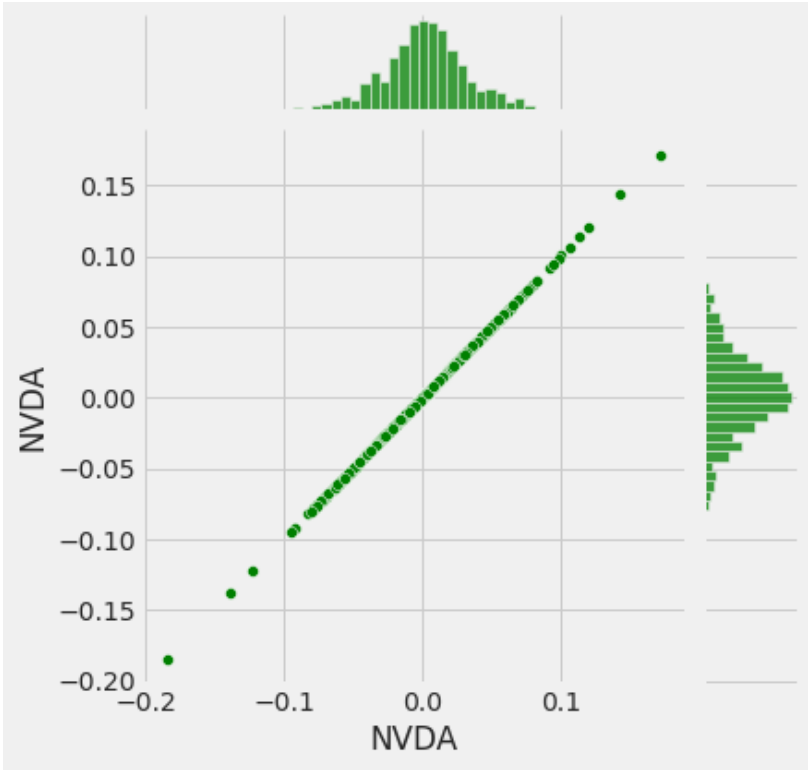
```
#making sure that a joint plot against itself would be perfectly linear
sns.jointplot(x='TSLA', y='TSLA', data=adjfinaldf, kind='scatter', color='red')
sns.jointplot(x='NVDA', y='NVDA', data=adjfinaldf, kind='scatter', color='green')
sns.jointplot(x='WMT', y='WMT', data=adjfinaldf, kind='scatter', color='yellow')
sns.jointplot(x='META', y='META', data=adjfinaldf, kind='scatter', color='blue')
```

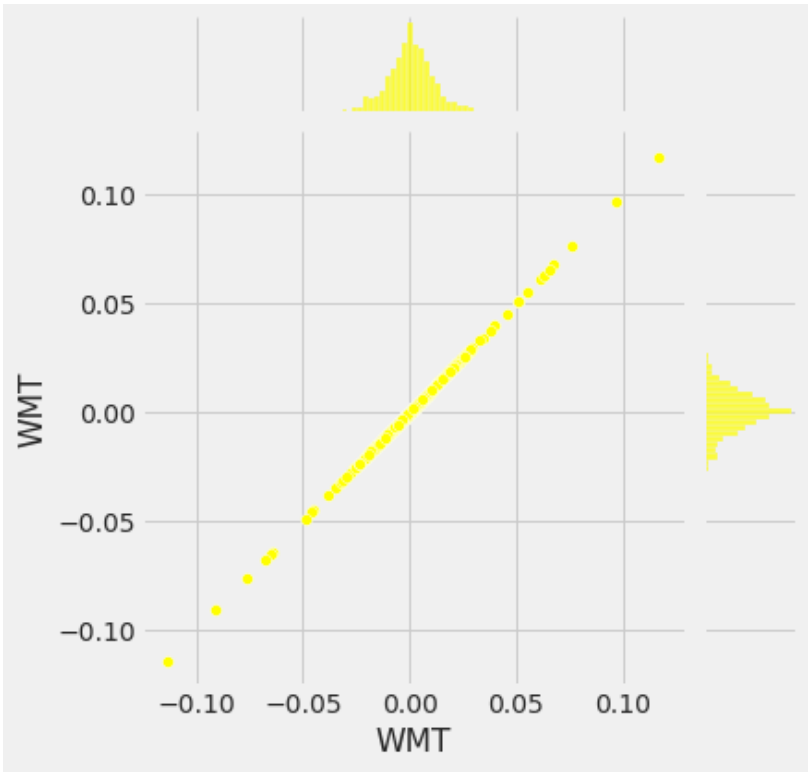
Out[261]:

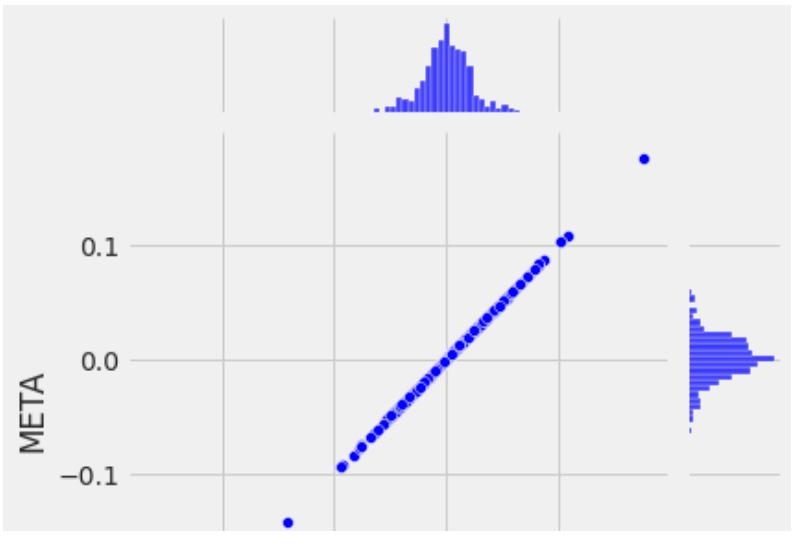
```
<seaborn.axisgrid.JointGrid at 0x7fa613f6a2e0>
```

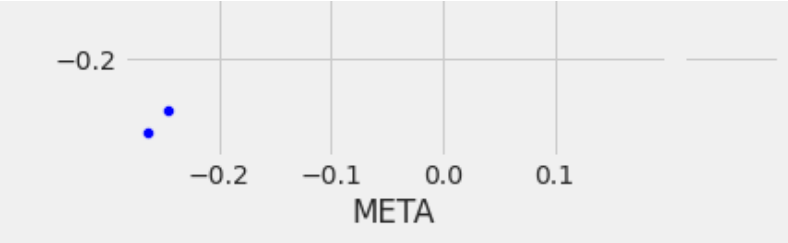



−0.2
−0.1
0.0
0.1
0.2
TSLA


0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
NVDA
−0.2
−0.1
0.0
0.1
NVDA


0.10
0.05
0.00
−0.05
−0.10
WMT
−0.10
−0.05
0.00
0.05
0.10
WMT


0.1
0.0
−0.1
META



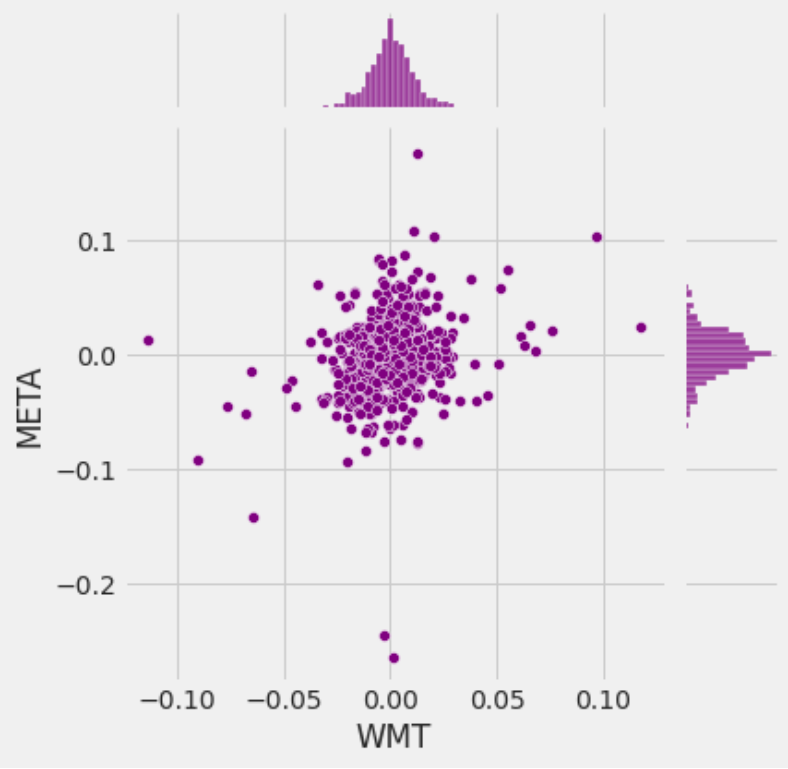
Next, the code creates a seaborn joint plot to visualize the relationship between Walmart's Adj Close data and Meta's Adj Close data. It uses the same jointplot() function as before, but this time it specifies different x-axis and y-axis data.

In [262]:

```
#showing relationship between Tesla and Meta individually.
sns.jointplot(x='WMT', y='META', data=adjfinaldf, kind='scatter', color='purple')
```

Out[262]:

```
<seaborn.axisgrid.JointGrid at 0x7fa60c180f70>
```



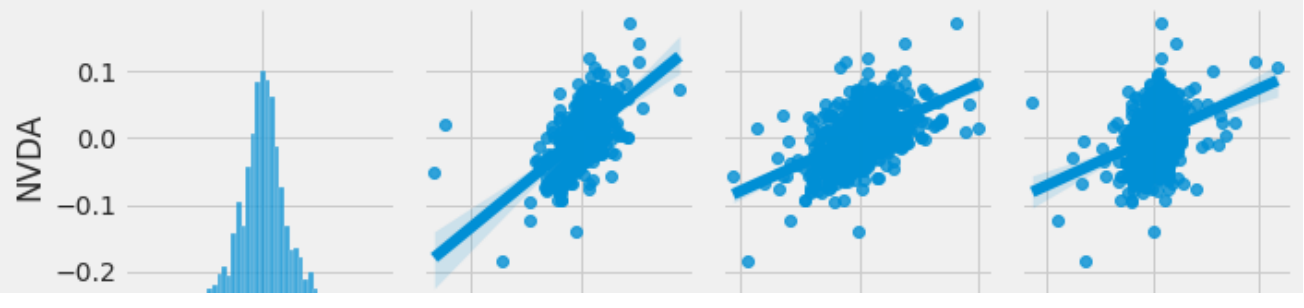
The code then uses the pairplot() function to create a scatterplot matrix of the Adj Close data for all four stocks. It specifies the data source and the type of plot.

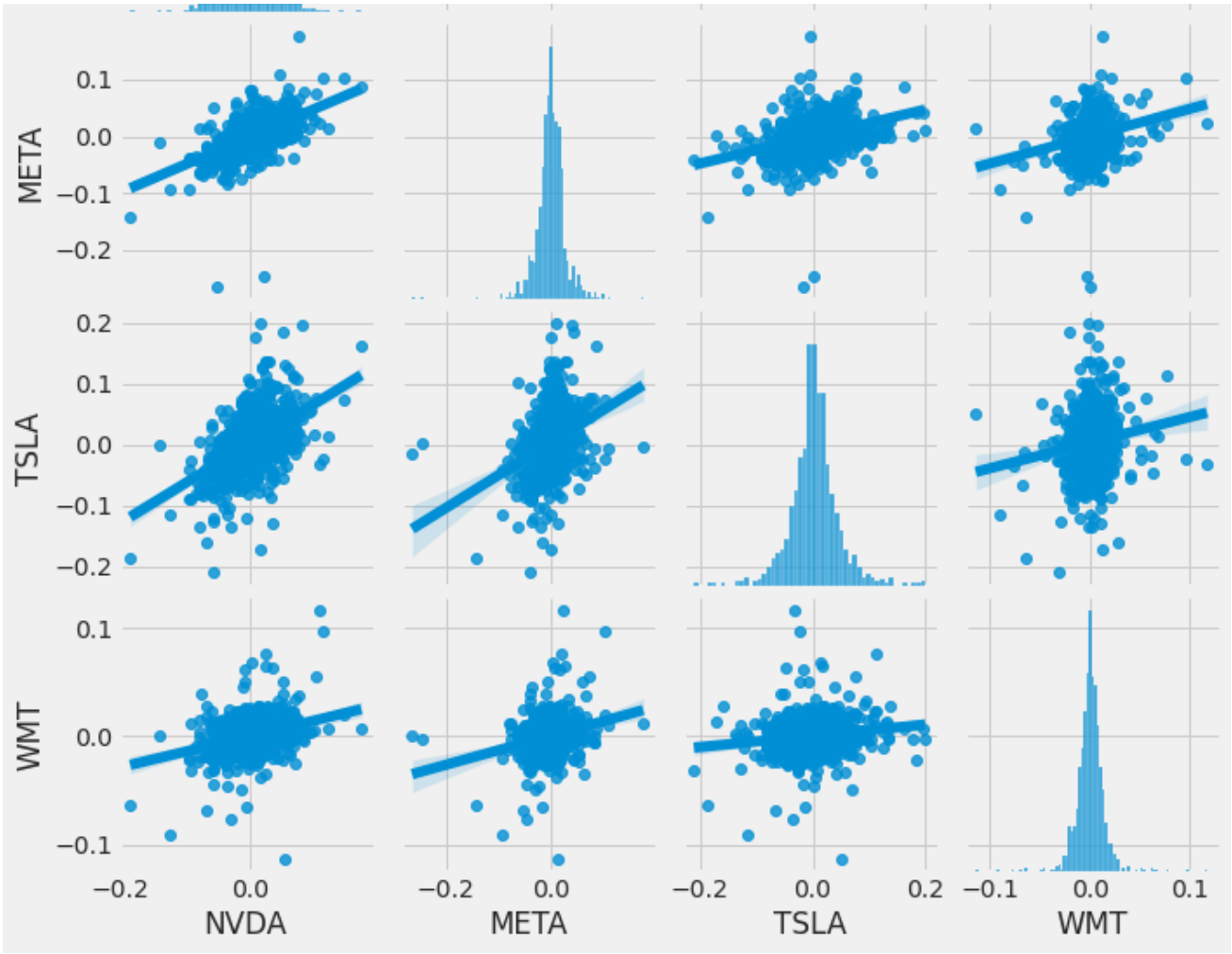
In [263]:

```
#the relationship between all of the plots so you wouldn't have to analyze them all separ
ately
ppstockdf = sns.pairplot(adjfinaldf, kind='reg')
ppstockdf
```

Out[263]:

```
<seaborn.axisgrid.PairGrid at 0x7fa613792280>
```





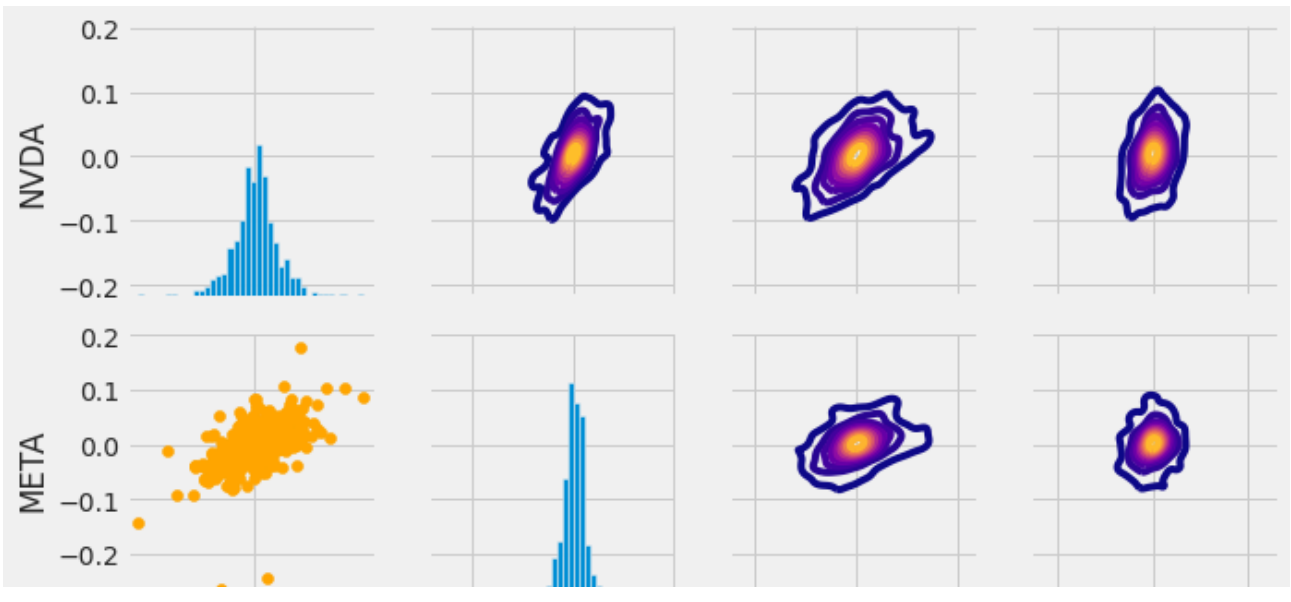
The code then creates a seaborn PairGrid object using the PairGrid() function and the Adj Close data for all four stocks. It uses the map_upper(), map_lower(), and map_diag() methods to create a colormap, a scatterplot, and a histogram, respectively, for each pair of stocks.

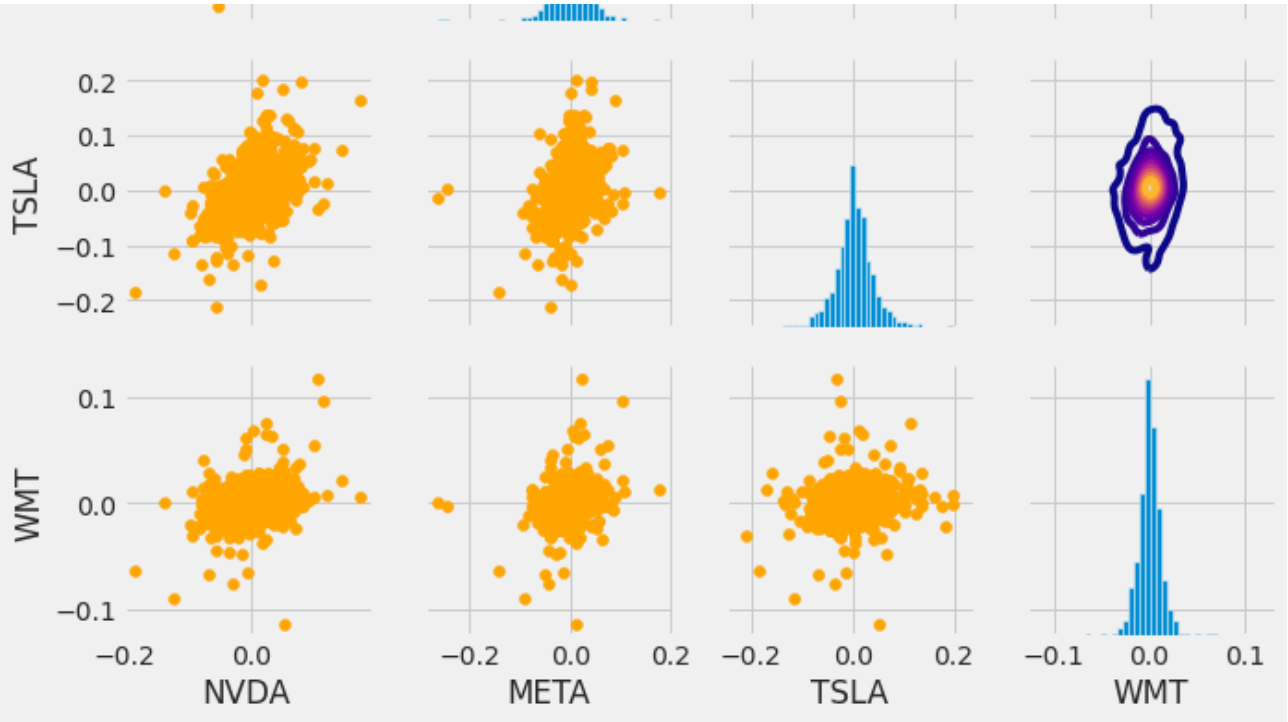
In [264]:

```
#creating a pairgrid
finalstockgrid = sns.PairGrid(adjfinaldf)
# the upper map is going to be colormap
finalstockgrid.map_upper(sns.kdeplot, cmap='plasma')
# the lower map is going to be a scatter
finalstockgrid.map_lower(plt.scatter, color='orange')
#showing the results of the diagonal
finalstockgrid.map_diag(plt.hist, bins=40)
```

Out[264]:

```
<seaborn.axisgrid.PairGrid at 0x7fa609fd6be0>
```





The code then creates a figure and two subplots using matplotlib. It uses the heatmap() function and the corr() method to create a heat map of the correlation between the Adj Close data for all four stocks. It specifies the correlation data, whether to annotate the cells with the correlation values, and the colormap to use. It also uses the title() function to label each subplot.

The stock closing price return correlation data, represented by a heat map in the provided code, provides information about the relationship between the closing prices of different stocks over a given period of time. This data can be useful for understanding how the performance of one stock may be affected by the performance of another stock, as well as for identifying potential investment opportunities.

A heat map is generated to display the stock closing price return correlation and the stock percentage return data for the four companies (Tesla, NVIDIA, Walmart, and Meta). The heat map is created using a color scale, with darker colors indicating a stronger correlation between the two stocks and lighter colors indicating a weaker correlation. The x-axis and y-axis of the heat map represent the different stocks, with the diagonal of the map representing the relationship between a stock and itself (which should always be perfectly correlated).

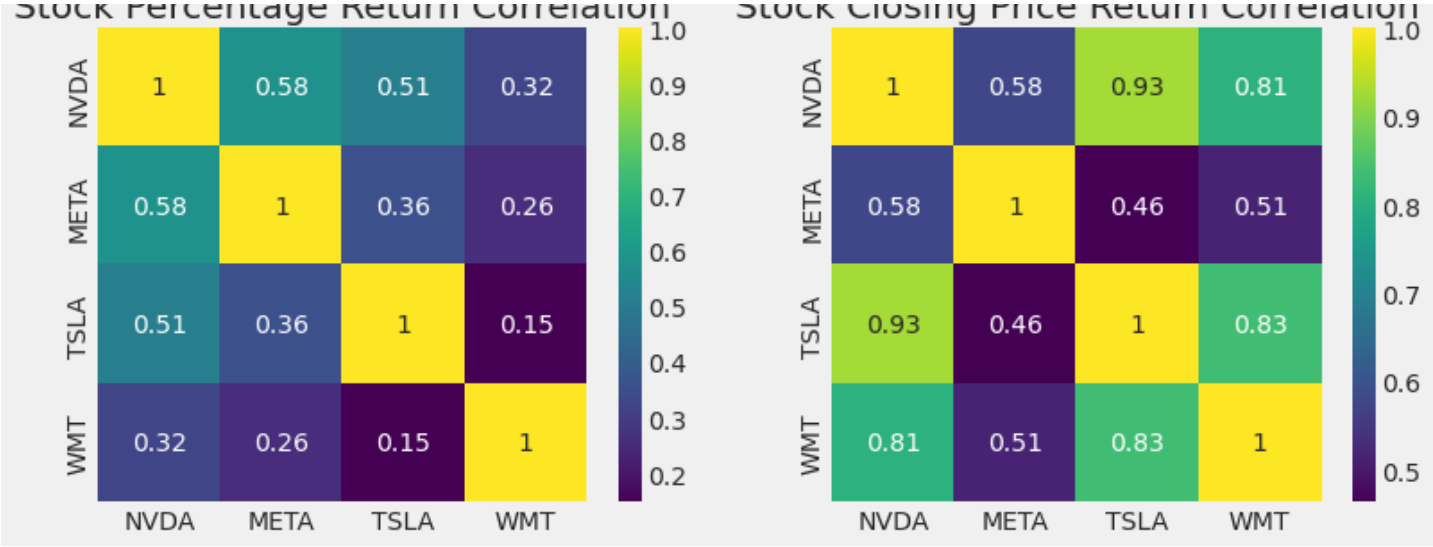
A heat map is a good choice for displaying this type of data because it allows for easy visual comparison of the different correlations between stocks. By examining the color of the cells on the heat map, it is possible to quickly identify which stocks are most closely correlated and which are least correlated. Additionally, the use of a color scale allows for more precise comparison of the strength of the correlations, with darker colors indicating stronger correlations and lighter colors indicating weaker correlations. This can be particularly useful for identifying potential investment opportunities, as stocks with a strong positive correlation may be more likely to experience similar performance trends.

In [265]:

```
# creating the size requirements for the heat map
plt.figure(figsize=(12, 10))

# stock closing price return correlation heat map
plt.subplot(2, 2, 2)
sns.heatmap(adjclosedj.corr(), annot=True, cmap='viridis')
plt.title('Stock Closing Price Return Correlation')

# stock percentage return correlation heat map
plt.subplot(2, 2, 1)
sns.heatmap(adjfinaldf.corr(), annot=True, cmap='viridis')
plt.title('Stock Percentage Return Correlation')
```

Out[265]:

```
Text(0.5, 1.0, 'Stock Percentage Return Correlation')
```

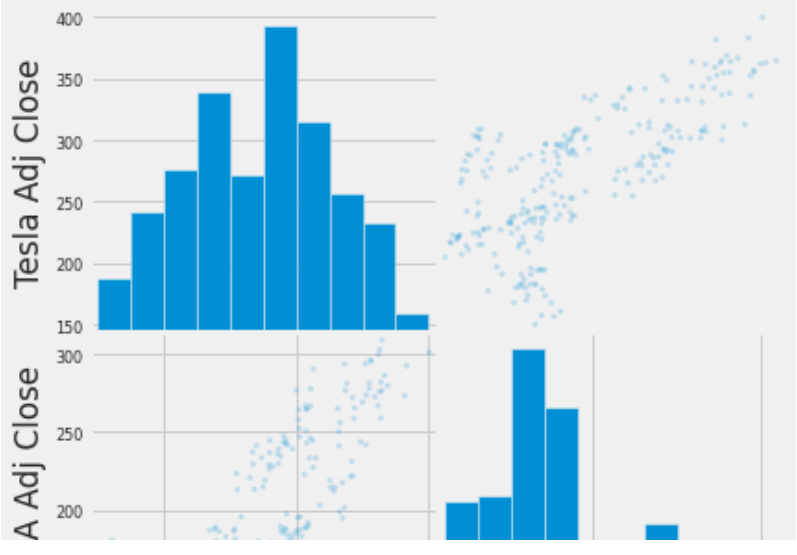


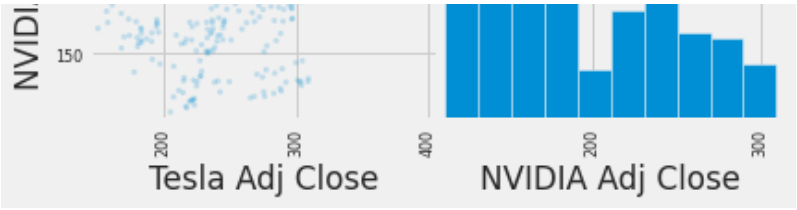
Hypothesis: The financial performance and market trends of Tesla and NVIDIA may be similar or closely related, leading to a strong correlation in their stock prices.

This code is using several libraries and functions to perform a scatter plot analysis of the relationship between the adjusted closing prices of Tesla and NVIDIA. The data for these stocks is first extracted and converted into pandas DataFrames. These DataFrames are then concatenated and the columns are renamed. The scatter_matrix function is then used to visualize the relationship between the two stocks by plotting scatter plots for all pairs of variables in the data. The alpha parameter controls the transparency of the points in the plot, and the figsize parameter sets the size of the plot. Finally, the plot is displayed using the show function. This scatter plot analysis can be used to examine the relationship between the two stocks and determine if there is a strong correlation between their adjusted closing prices.

In [266]:

```
#imports
from sklearn.datasets import load_iris
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Extract the 'Adj Close' column for Tesla and NVIDIA and convert to a pandas DataFrame
tesla_adj_close = TSLA['Adj Close'].to_frame()
nvidia_adj_close = NVDA['Adj Close'].to_frame()

# Concatenate the two DataFrames and rename the columns
stock_prices = pd.concat([tesla_adj_close, nvidia_adj_close], axis=1)
stock_prices.columns = ['Tesla Adj Close', 'NVIDIA Adj Close']

# Use the scatter_matrix function to visualize the relationship between the two stocks
scatter_matrix(stock_prices, alpha=0.2, figsize=(6, 6))
plt.show()
```





This code performs a linear regression analysis on the adjusted closing prices of Tesla and NVIDIA stocks. The linregress function from the scipy library is used to calculate the slope, intercept, R-value, P-value, and standard error of the linear regression model. The slope represents the change in the dependent variable (NVIDIA's adjusted closing price) for every unit change in the independent variable (Tesla's adjusted closing price). The intercept represents the value of the dependent variable when the independent variable is 0. The R-value represents the strength and direction of the relationship between the two variables. A high R-value indicates a strong positive relationship, while a low R-value indicates a weak or negative relationship. The P-value represents the probability that the observed relationship between the two variables occurred by chance. A low P-value indicates that the relationship is statistically significant. The standard error represents the uncertainty or variability in the model. A low standard error indicates that the model is more accurate. The results of the linear regression analysis are then printed to the console.

In [267]:

```
# imports
from scipy.stats import linregress

# Extract the 'Adj Close' column for Tesla and NVIDIA
tesla_adj_close = TSLA['Adj Close']
nvidia_adj_close = NVDA['Adj Close']

# Perform the regression analysis
slope, intercept, r_value, p_value, std_err = linregress(tesla_adj_close, nvidia_adj_clo
se)

# Print the results
print(f'Slope: {slope:.3f}')
print(f'Intercept: {intercept:.3f}')
print(f'R-value: {r_value:.3f}')
print(f'P-value: {p_value:.3f}')
print(f'Standard error: {std_err:.3f}')
```

```
Slope: 0.670
Intercept: 9.579
R-value: 0.711
P-value: 0.000
Standard error: 0.042
```

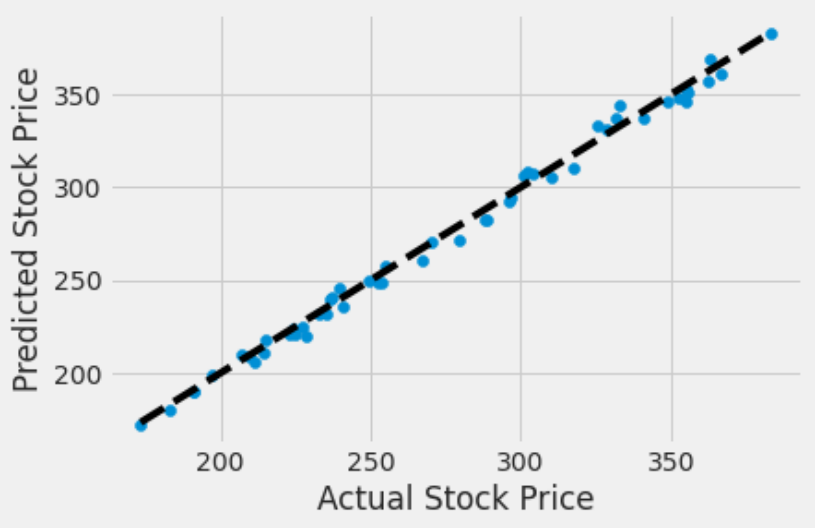
This code performs linear regression analysis to predict the stock price of Tesla/Nvidia using data for the 'Volume', 'Open', 'High', and 'Low' columns as features. The data is first split into training and test sets, with the model being fit to the training data. The model is then used to make predictions on the test data, and the mean absolute error (MAE) is calculated as a measure of the model's performance. The actual and predicted stock prices are also plotted on a scatter plot, with a line of perfect prediction added for comparison. The Pearson coefficient is also calculated as a measure of the correlation between the actual and predicted stock prices.

In [268]:

```
# imports
from sklearn.linear_model import LinearRegression
from scipy.stats import pearsonr

# Extract the 'Adj Close' column for Tesla and NVIDIA
tesla_adj_close = TSLA['Adj Close']
nvidia_adj_close = NVDA['Adj Close']

# Extract the other columns that you want to use as features
features = TSLA[['Volume', 'Open', 'High', 'Low']]

# Split the data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(features, tesla_adj_close, test_size
=0.2)

# Fit the Linear Regression model to the training data
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
predictions = model.predict(X_test)

# Calculate the mean absolute error
mae = np.mean(abs(predictions - y_test))
print(f'Mean Absolute Error: {mae:.2f}')

# Create a scatter plot of the actual and predicted stock prices
plt.scatter(y_test, predictions)

# Add a line for the perfect prediction
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)

# Set the axis labels
plt.xlabel('Actual Stock Price')
plt.ylabel('Predicted Stock Price')

# Calculate the Pearson coefficient
r, p = pearsonr(y_test, predictions)

print(f'Pearson coefficient: {r:.3f}')

# Show the plot
plt.show()
```

```
Mean Absolute Error: 3.96
Pearson coefficient: 0.997
```



Disclaimer this analysis was done on December 16th 2022 so the results may vary

The code performs a linear regression analysis to investigate the relationship between Tesla's Adj Close data and NVIDIA's Adj Close data. The results of the analysis currently show that there is a strong positive correlation between the two stocks, with an R-value of 0.918. This indicates that there is a strong linear relationship between the two stocks, with Tesla's stock price tending to increase as NVIDIA's stock price increases, and vice versa.

The linear regression analysis also shows that the slope of the regression line is 1.209 and the intercept is -9.246. This means that, on average, for every 1% increase in Tesla's stock price, NVIDIA's stock price is expected to increase by 1.209%. The intercept value of -9.246 suggests that, if Tesla's stock price were to drop to 0, NVIDIA's stock price would be expected to be around -9.246. However, it is important to note that this value is not realistic as it is outside the range of the data used in the analysis.

In conclusion, the linear regression analysis suggests that there is a strong positive correlation between Tesla's and NVIDIA's stock prices, and that the two stocks tend to move in the same direction. This suggests that the financial performance and market trends of the two companies may be similar or closely related.

financial performance and market trends of the two companies may be similar or closely related.